6 Securing your system: IAM, security groups, and VPC

This chapter covers

- Who is responsible for security?
- Keeping your software up to date
- Controlling access to your AWS account with users and roles
- Keeping your traffic under control with security groups
- Using CloudFormation to create a private network

If security is a wall, you'll need a lot of bricks to build that wall as shown in figure 6.1. This chapter focuses on the four most important bricks to secure your systems on AWS:

1. *Installing software updates*—New security vulnerabilities are found in software every day. Software vendors release updates to fix those vulnerabilities and it's your job to install those updates as quickly as possible after they're released. Otherwise your system will be an easy victim for hackers.

2. *Restricting access to your AWS account*—This becomes even more important if you aren't the only one accessing your AWS account (if coworkers and scripts are also accessing it). A buggy script could easily terminate all your EC2 instances instead of only the one you intended. Granting only the permissions you need is key to securing your AWS resources from accidental or intended disastrous actions.
3. *Controlling network traffic to and from your EC2 instances*—You only want ports to be accessible if they must be. If you run a web server, the only ports you need to open to the outside world are port 80 for HTTP traffic and 443 for HTTPS traffic. Close down all the other ports!
4. *Creating a private network in AWS*—You can create subnets that aren't reachable from the internet. And if they're not reachable, nobody can access them. Really, nobody? You'll learn how you can get access to them while preventing others from doing so.

One important brick is missing: securing your applications. We do not cover application security in our book. When buying or developing applications, you should follow security standards. For example, you need to check user input and allow only the necessary characters, don't save passwords in plain text, and use TLS/SSL to encrypt traffic between your virtual machines and your users. If you are installing applications with a package manager for your operating system, using Amazon Inspector (https://aws.amazon.com/inspector/) allows you to run automated security assessments.

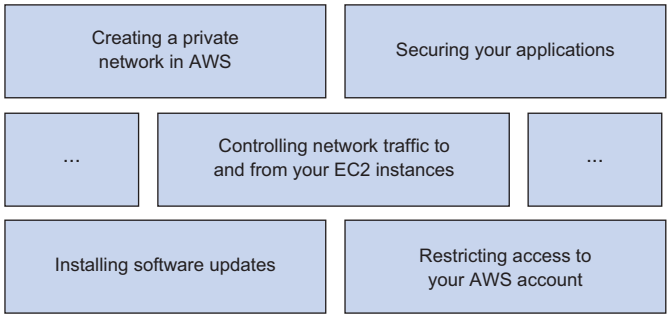


Figure 6.1 To achieve security of your cloud infrastructure and application, all security building blocks have to be in place.

Not all examples are covered by Free Tier

The examples in this chapter are *not* all covered by the Free Tier. A special warning message appears when an example incurs costs. As for the other examples, as long as you don't run them longer than a few days, you won't pay anything for them. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

Chapter requirements

To fully understand this chapter, you should be familiar with the following concepts:

- Subnet
- Route tables
- Access control lists (ACLs)
- Gateway
- Firewall
- Port
- Access management
- Basics of the Internet Protocol (IP), including IP addresses

Before we look at the four bricks, let's talk about how responsibility is divided between you and AWS.

6.1 Who's responsible for security?

The cloud is a shared-responsibility environment, meaning responsibility is shared between you and AWS. AWS is responsible for the following:

- Protecting the network through automated monitoring systems and robust internet access, to prevent Distributed Denial of Service (DDoS) attacks.
- Performing background checks on employees who have access to sensitive areas.
- Decommissioning storage devices by physically destroying them after end of life.
- Ensuring the physical and environmental security of data centers, including fire protection and security staff.

The security standards are reviewed by third parties; you can find an up-to-date overview at http://aws.amazon.com/compliance/.

What are your responsibilities?

- Implementing access management that restricts access to AWS resources like S3 and EC2 to a minimum, using AWS IAM.
- Encrypting network traffic to prevent attackers from reading or manipulating data (for example, using HTTPS).
- Configuring a firewall for your virtual network that controls incoming and outgoing traffic with security groups and ACLs.
- Encrypting data at rest. For example, enable data encryption for your database or other storage systems.
- Managing patches for the OS and additional software on virtual machines.

Security involves an interaction between AWS and you, the customer. If you play by the rules, you can achieve high security standards in the cloud.

6.2 Keeping your software up to date

Not a week goes by without the release of an important update to fix security vulnerabilities in some piece of software or another. Sometimes your OS is affected, or software libraries like OpenSSL. Other times it's an environments like Java, Apache, and PHP; or an application like WordPress. If a security update is released, you must install it quickly, because the exploit may have already been released, or because unscrupulous people could look at the source code to reconstruct the vulnerability. You should have a working plan for how to apply updates to all running virtual machines as quickly as possible.

6.2.1 Checking for security updates

If you log in to an Amazon Linux EC2 instance via SSH, you'll see message like the following:

```
$ ssh ec2-user@ec2-34-230-84-110.compute-1.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
8 package(s) needed for security, out of 8 available
Run "sudo yum update" to apply all updates.
```

Eight security updates are available.

This example shows that eight security updates are available; this number will vary when you look for updates. AWS won't apply updates for you on your EC2 instances—you're responsible for doing so.

You can use the `yum` package manager to handle updates on Amazon Linux. Run `yum --security check-update` to see which packages require a security update:

The output will be different when you run the command.

```
$ yum --security check-update
Loaded plugins: priorities, update-motd, upgrade-helper
8 package(s) needed for security, out of 8 available

authconfig.x86_64      6.2.8-30.31.amzn1    amzn-updates
bash.x86_64            4.2.46-28.37.amzn1   amzn-updates
curl.x86_64            7.51.0-9.75.amzn1    amzn-updates
glibc.x86_64           2.17-196.172.amzn1   amzn-updates
glibc-common.x86_64    2.17-196.172.amzn1   amzn-updates
kernel.x86_64          4.9.43-17.38.amzn1   amzn-updates
libcurl.x86_64         7.51.0-9.75.amzn1    amzn-updates
wget.x86_64            1.18-3.27.amzn1      amzn-updates
```

These packages are installed by default. Updates are available fixing security issues.

We encourage you to subscribe to the feed at the Amazon Linux AMI Security Center at https://alas.aws.amazon.com to receive security bulletins affecting Amazon Linux. Whenever a new security update is released, you should check whether you're affected

and act accordingly. If you are using another Linux distribution or operating system, you should follow the relevant security bulletins.

When dealing with security updates, you may face either of these situations:

- When the virtual machine starts the first time, many security updates may need to be installed in order for the machine to be up to date.
- New security updates are released when your virtual machine is running, and you need to install these updates while the machine is running.

Let's look how to handle these situations.

6.2.2 Installing security updates on startup

If you create your EC2 instances with CloudFormation templates, you have three options to install security updates on startup:

1. *Install all updates at the end of the boot process* by including `yum -y update` in your user-data script.
2. *Install security updates at the end of the boot process only* by including `yum -y --security update` in your user-data script.
3. *Define the package versions explicitly*. Install updates identified by a version number.

The first two options can be easily included in the user data of your EC2 instance. You can find the code in `/chapter06/ec2-yum-update.yaml` in the book's code folder. You install all updates as follows:

```
Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      # [...]
      UserData:
        'Fn::Base64': |
          #!/bin/bash -x
          yum -y update      <-- Installs all updates
```

To install only security updates, do the following:

```
Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      # [...]
      UserData:
        'Fn::Base64': |
          #!/bin/bash -x
          yum -y --security update        <-- Installs only security updates
```

The problem with installing all updates is that your system becomes unpredictable. If your virtual machine was started last week, all updates that were available last week have been applied. But in the meantime, new updates have likely been released. If you start a new VM today and install all updates, you'll end up with a different machine

than the machine from last week. *Different* can mean that for some reason it's no longer working. That's why we encourage you to explicitly define and test the updates you want to install. To install security updates with an explicit version, you can use the `yum update-to` command. `yum update-to` updates a package to an explicit version instead of the latest:

```
yum update-to bash-4.2.46-28.37.amzn1
```

Updates Bash to version 4.2.46-28.37.amznl

Using a CloudFormation template to describe an EC2 instance with explicitly defined updates looks like this:

```
Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      # [...]
      UserData:
        'Fn::Base64': |
          #!/bin/bash -x
          yum update-to bash-4.2.46-28.37.amzn1
```

The same approach works for non-security-related package updates. Whenever a new security update is released, you should check whether you're affected and modify the user data to keep new systems secure.

6.2.3 Installing security updates on running virtual machines

What do you do if you have to install a security update of a core component on tens or even hundreds of virtual machines? You could manually log in to all your VMs using SSH and run `yum -y --security update` or `yum update-to [...]`, but if you have many machines or the number of machines grows, this can be annoying. One way to automate this task is to use a small script that gets a list of your VMs and executes `yum` in all of them. The following listing shows how this can be done in with the help of a Bash script. You can find the code in `/chapter06/update.sh` in the book's code folder.

Listing 6.1 Installing security updates on all running EC2 instances

```
PUBLICIPADDRESSES="$(aws ec2 describe-instances \
➥ --filters "Name=instance-state-name,Values=running" \
➥ --query "Reservations[].Instances[].PublicIpAddress" \
➥ --output text)"

for PUBLICIPADDRESS in $PUBLICIPADDRESSES; do
  ssh -t "ec2-user@$PUBLICIPADDRESS" \
➥ "sudo yum -y --security update"
done
```

Gets all public names of running EC2 instances

Connects via SSH

Executes the yum update command

Now you can quickly apply updates to all of your running machines.

AWS Systems Manager: apply patches in an automated way

Using SSH to install security updates on all your virtual machines is challenging. You need to have a network connection as well as a key for each virtual machine. Handling errors during applying patches is another challenge.

The AWS Systems Manager service is a powerful tool when managing virtual machines. First you install an agent on each virtual machine. You then control your EC2 instances with the help of AWS SSM, for example by creating a job to patch all your EC2 instances with the latest patch level from the AWS Management Console.

Some security updates require you to reboot the VM—for example, if you need to patch the kernel of your VMs running on Linux. You can automate the reboot of the machines or switch to an updated AMI and start new virtual machines instead. For example, new AMIs of Amazon Linux including the latest packages are released frequently.

6.3 Securing your AWS account

Securing your AWS account is critical. If someone gets access to your AWS account, they can steal your data, use resources at your expense, or delete all your data. As figure 6.2 shows, an AWS account is a basket for all the resources you own: EC2 instances, CloudFormation stacks, IAM users, and so on. Each AWS account comes with a root user granted unrestricted access to all resources. So far, you've used the root user to log into the Management Console and the user `mycli`—created in section 4.2—when using the CLI. In this section you will create an additional user to log into the Management Console, to avoid using the root user at all. Doing so allows you to manage multiple users, each restricted to the resources that are necessary for their roles.

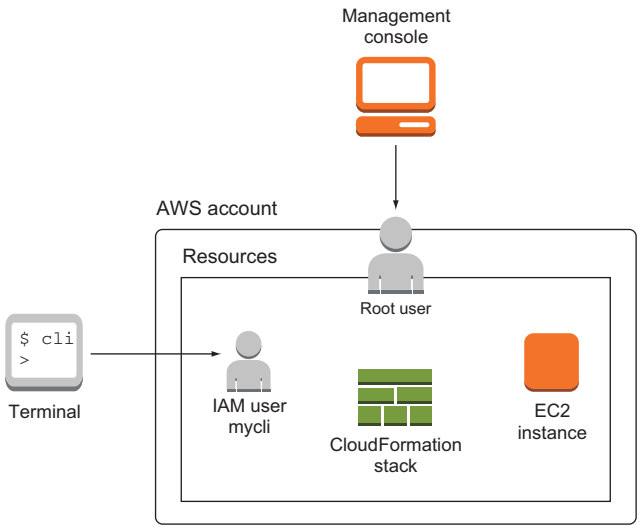


Figure 6.2 An AWS account contains all the AWS resources and comes with a root user by default.

To access your AWS account, an attacker must be able to authenticate to your account. There are three ways to do so: using the root user, using a normal user, or authenticating as an AWS resource like an EC2 instance. To authenticate as a root user or normal user, the attacker needs the username and password or the access keys. To authenticate as an AWS resource like an EC2 instance, the attacker needs to send API/CLI requests from that machine.

To protect yourself from an attacker stealing or cracking your passwords or access keys, you will enable multi-factor authentication (MFA) for your root user, to add an additional layer of security to the authentication process, in the following section.

6.3.1 Securing your AWS account's root user

We advise you to enable MFA for the root user of your AWS account. After MFA is activated, a password and a temporary token are needed to log in as the root user.

Follow these steps to enable MFA, as shown in figure 6.3:

1. Click your name in the navigation bar at the top of the Management Console.
2. Select My Security Credentials.
3. A pop-up should appear. Click Continue to Security Credentials.
4. Install an MFA app on your smartphone, one that supports the TOTP standard (such as Google Authenticator).
5. Expand the Multi-factor authentication (MFA) section.
6. Click Activate MFA.
7. Select a virtual MFA device and proceed with the next step.
8. Follow the instructions in the wizard. Use the MFA app on your smartphone to scan the QR code that is displayed.

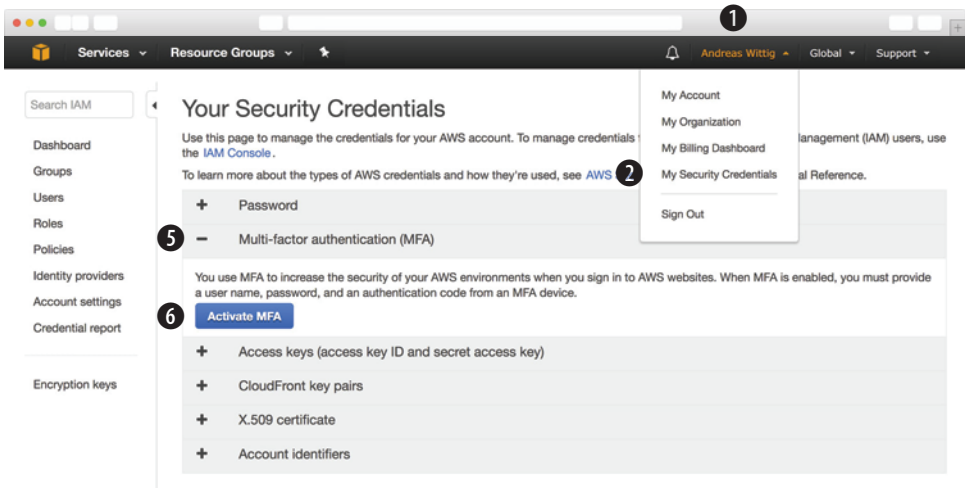


Figure 6.3 Protect your root user with multi-factor authentication (MFA).

If you're using your smartphone as a virtual MFA device, it's a good idea not to log in to the Management Console from your smartphone or to store the root user's password on the phone. Keep the MFA token separate from your password.

6.3.2 AWS Identity and Access Management (IAM)

Figure 6.4 shows an overview of all the core concepts of the Identity and Access Management (IAM) service. This service provides authentication and authorization for the AWS API. When you send a request to the AWS API, IAM verifies your identity and checks if you are allowed to perform the action. IAM controls who (authentication) can do what (authorization) in your AWS account. For example, is the user allowed to launch a new virtual machine?

- An *IAM user* is used to authenticate people accessing your AWS account.
- An *IAM group* is a collection of IAM users.
- An *IAM role* is used to authenticate AWS resources, for example an EC2 instance.
- An *IAM policy* is used to define the permissions for a user, group, or role.

Table 6.1 shows the differences between users and roles. Roles authenticate AWS entities such as EC2 instances. IAM users authenticate the people who manage AWS resources, for example system administrators, DevOps engineers, or software developers.

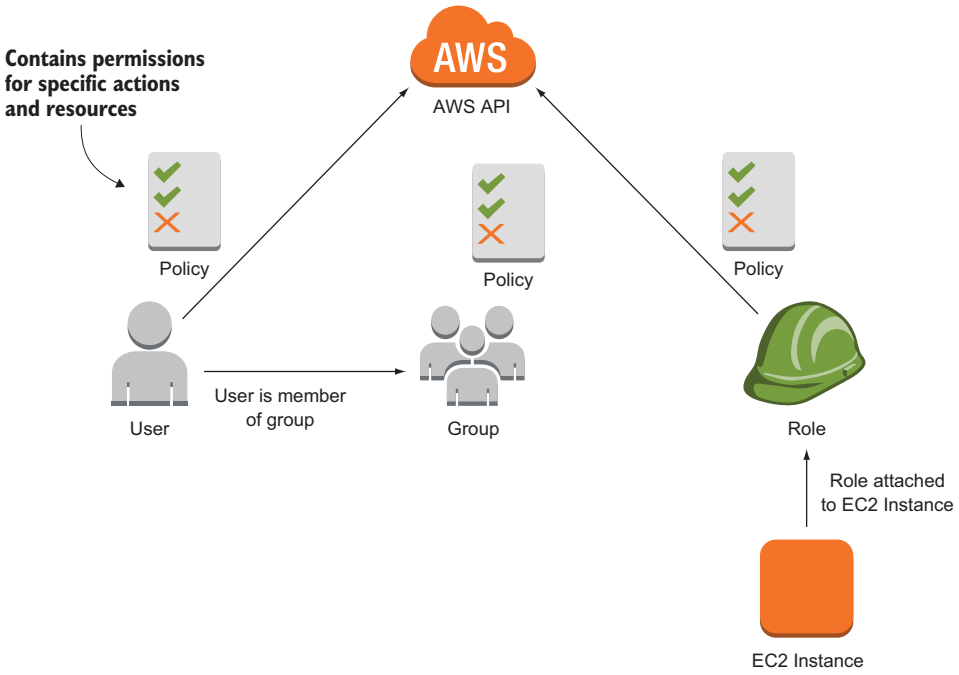


Figure 6.4 IAM concepts

Table 6.1 Differences between root user, IAM user, and IAM role

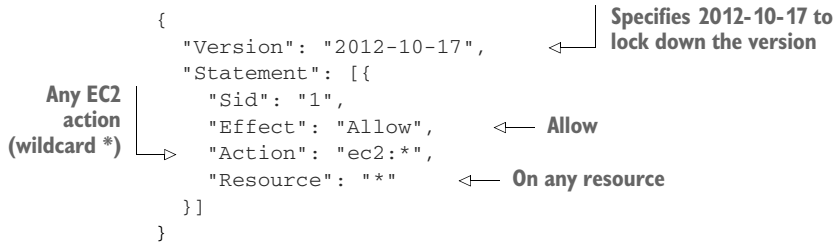
	Root user	IAM user	IAM role
Can have a password (needed to log into the AWS Management Console)	Always	Yes	No
Can have access keys (needed to send requests to the AWS API (for example, for CLI or SDK)	Yes (not recommended)	Yes	No
Can belong to a group	No	Yes	No
Can be associated with an EC2 instance	No	No	Yes

By default, users and roles can't do anything. You have to create a policy stating what actions they're allowed to perform. IAM users and IAM roles use policies for authorization. Let's look at policies first.

6.3.3 Defining permissions with an IAM policy

By attaching one or multiple IAM policies to an IAM user or role, you are granting permissions to manage AWS resources. Policies are defined in JSON and contain one or more statements. A statement can either allow or deny specific actions on specific resources. The wildcard character * can be used to create more generic statements.

The following policy has one statement that allows every action for the EC2 service, for all resources:



If you have multiple statements that apply to the same action, `Deny` overrides `Allow`. The following policy allows all EC2 actions except terminating EC2 instances:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "1",
    "Effect": "Allow",
    "Action": "ec2:*",
    "Resource": "*"
  }, {
    "Sid": "2",
    "Effect": "Deny",
    "Action": "ec2:TerminateInstances",
    "Resource": "*"
  }]
}
```

Deny

Terminating EC2 instances

The following policy denies all EC2 actions. The `ec2:TerminateInstances` statement isn't crucial, because `Deny` overrides `Allow`. When you deny an action, you can't allow that action with another statement:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "1",
    "Effect": "Deny",
    "Action": "ec2:*",
    "Resource": "*"
  }, {
    "Sid": "2",
    "Effect": "Allow",
    "Action": "ec2:TerminateInstances",
    "Resource": "*"
  }]
}
```

Denies every EC2 action

Allow isn't crucial; Deny overrides Allow.

So far, the `Resource` part has been `["*"]` for every resource. Resources in AWS have an Amazon Resource Name (ARN); figure 6.5 shows the ARN of an EC2 instance.

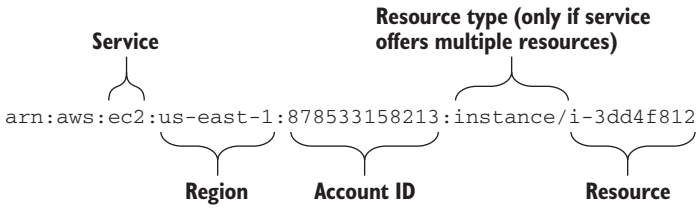


Figure 6.5 Components of an Amazon Resource Name (ARN) identifying an EC2 instance

To find out the account ID, you can use the CLI:

```
$ aws iam get-user --query "User.Arn" --output text
arn:aws:iam::111111111111:user/mycli
```

Account ID has 12 digits. 111111111111 in our example.

If you know your account ID, you can use ARNs to allow access to specific resources of a service:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "2",
    "Effect": "Allow",
    "Action": "ec2:TerminateInstances",
    "Resource":
➥ "arn:aws:ec2:us-east-1:111111111111:instance/i-0b5c991e026104db9"
  }]
}
```

There are two types of policies:

1. *Managed policy*—If you want to create policies that can be reused in your account, a managed policy is what you're looking for. There are two types of managed policies:
   - *AWS managed policy*—A policy maintained by AWS. There are policies that grant admin rights, read-only rights, and so on.
   - *Customer managed*—A policy maintained by you. It could be a policy that represents the roles in your organization, for example.
2. *Inline policy*—A policy that belongs to a certain IAM role, user, or group. An inline policy can't exist without the IAM role, user, or group that it belongs to.

With CloudFormation, it's easy to maintain inline policies; that's why we use inline policies most of the time in this book. One exception is the `mycli` user: this user has the AWS managed policy `AdministratorAccess` attached. We maintain an open-source list of all possible IAM permissions available at https://iam.cloudonaut.io/.

6.3.4 Users for authentication, and groups to organize users

A user can authenticate using either a user name and password, or access keys. When you log in to the Management Console, you're authenticating with your user name and password. When you use the CLI from your computer, you use access keys to authenticate as the `mycli` user.

You're using the root user at the moment to log in to the Management Console. You should create an IAM user instead, for two reasons.

- Creating IAM users allows you to set up a unique user for every person who needs to access your AWS account.
- You can grant access only to the resources each user needs, allowing you to follow the least privilege principle.

To make things easier if you want to add users in the future, you'll first create a group for all users with administrator access. Groups can't be used to authenticate, but they centralize authorization. So, if you want to stop your admin users from terminating EC2 instances, you only need to change the policy for the group instead of changing it for all admin users. A user can be a member of zero, one, or multiple groups.

It's easy to create groups and users with the CLI. Replace *`$Password`* in the following with a secure password:

```
$ aws iam create-group --group-name "admin"
$ aws iam attach-group-policy --group-name "admin" \
➥ --policy-arn "arn:aws:iam::aws:policy/AdministratorAccess"
$ aws iam create-user --user-name "myuser"
$ aws iam add-user-to-group --group-name "admin" --user-name "myuser"
$ aws iam create-login-profile --user-name "myuser" --password "$Password"
```

The user `myuser` is ready to be used. But you must use a different URL to access the Management Console if you aren't using the root user: https://$accountId.signin.aws.amazon.com/console. Replace `$accountId` with the account ID that you extracted earlier with the `aws iam get-user` command.

Enabling MFA for IAM users

We encourage you to enable MFA for all users. If possible, don't use the same MFA device for your root user that you use for everyday users. You can buy hardware MFA devices for $13 from AWS partners like Gemalto. To enable MFA for your users, follow these steps:

- Open the IAM service in the Management Console.
- Choose Users at left.
- Select the `myuser` user.
- Select the Security credentials tab.
- Click the pencil near to Assigned MFA device. The wizard to enable MFA for the IAM user is the same one you used for the root user.

We do recommend enabling MFA for all users, especially for users granted administrator access to all or some services.

WARNING Stop using the root user from now on. Always use `myuser` and the new link to the Management Console.

WARNING You should never copy a user's access keys to an EC2 instance; use IAM roles instead! Don't store security credentials in your source code. And never ever check them into your Git or SVN repository. Try to use IAM roles instead whenever possible.

6.3.5 Authenticating AWS resources with roles

There are various use cases where an EC2 instance needs to access or manage AWS resources. For example, an EC2 instance might need to :

- Back up data to the object store S3.
- Terminate itself after a job has been completed.
- Change the configuration of the private network environment in the cloud.

To be able to access the AWS API, an EC2 instance needs to authenticate itself. You could create an IAM user with access keys and store the access keys on an EC2 instance for authentication. But doing so is a hassle, especially if you want to rotate the access keys regularly.

Instead of using an IAM user for authentication, you should use an IAM role whenever you need to authenticate AWS resources like EC2 instances. When using an IAM role, your access keys are injected into your EC2 instance automatically.

If an IAM role is attached to an EC2 instance, all policies attached to those roles are evaluated to determine whether the request is allowed. By default, no role is attached to an EC2 instance and therefore the EC2 instance is not allowed to make any calls to the AWS API.

The following example will show you how to use an IAM role for an EC2 instance. Do you remember the temporary EC2 instances from chapter 4? What if we forgot to terminate those VMs? A lot of money was wasted because of that. You'll now create an EC2 instance that stops itself automatically. The following snippet shows a one-liner terminating an EC2 instance after 5 minutes. The command `at` is used to execute the `aws ec2 stop-instances` with a 5 minute delay:

```
echo "aws ec2 stop-instances --instance-ids i-0b5c991e026104db9" \
➥ | at now + 5 minutes
```

The EC2 instance needs permission to stop itself. Therefore, you need to attach an IAM role to the EC2 instance. The role contains an inline policy granting access to the `ec2:StopInstances` action. The following code shows how you define an IAM role with the help of CloudFormation:

```
Role:
  Type: 'AWS::IAM::Role'
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
      - Effect: Allow
        Principal:
          Service: 'ec2.amazonaws.com'
        Action:
        - 'sts:AssumeRole'
    Policies:
    - PolicyName: ec2
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Sid: Stmt1425388787000
          Effect: Allow
          Action: 'ec2:StopInstances'
          Resource: '*'
          Condition:
            StringEquals:
              'ec2:ResourceTag/aws:cloudformation:stack-id':
➥ !Ref 'AWS::StackId'
```

Allow the EC2 service to assume this role.

Policies begin

Policy definition

Condition can solve the problem: only allow if tagged with the stack ID.

To attach an inline role to an instance, you must first create an instance profile:

```
InstanceProfile:
  Type: 'AWS::IAM::InstanceProfile'
  Properties:
    Roles:
    - !Ref Role
```

The following listing shows how to attach the IAM role to the virtual machine:

```
Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
    # [...]
    IamInstanceProfile: !Ref InstanceProfile
    UserData:
      'Fn::Base64': !Sub |
        #!/bin/bash -x
        INSTANCEID="$(curl -s http://169.254.169.254/\
➥ latest/meta-data/instance-id)"
        echo "aws ec2 stop-instances --instance-ids $INSTANCEID \
➥ --region ${AWS::Region}" | at now + ${Lifetime} minutes
```

Create the CloudFormation stack with the template located at http://mng.bz/Z35X by clicking on the CloudFormation Quick-Create link: http://mng.bz/833J. Specify the lifetime of the EC2 instance via the parameter, and pick the default VPC and subnet as well. Wait until the amount of time specified as the lifetime has passed, and see if your EC2 instance is stopped in the EC2 Management Console. The lifetime begins when the server is fully started and booted.

Cleaning up

Don't forget to delete your stack `ec2-iamrole` after you finish this section, to clean up all used resources. Otherwise you'll likely be charged for the resources you use (even when your EC2 instance is stopped, you pay for the network attached storage).

You have learned how to use IAM users to authenticate people and IAM roles to authenticate EC2 instances or other AWS resources. You've also seen how to grant access to specific actions and resources by using an IAM policy. The next section will cover controlling network traffic to and from your virtual machine.

6.4 Controlling network traffic to and from your virtual machine

You only want traffic to enter or leave your EC2 instance if it has to do so. With a firewall, you control ingoing (also called *inbound* or *ingress*) and outgoing (also called *outbound* or *egress*) traffic. If you run a web server, the only ports you need to open to the outside world are port 80 for HTTP traffic and 443 for HTTPS traffic. All other ports should be closed down. You should only open ports that must be accessible, just as you grant only the permissions you need with IAM. If you are using a firewall that allows only legitimate traffic, you close a lot of possible security holes. You can also prevent yourself from human failure, for example you prevent accidentally sending email to customers from a test system by not opening outgoing SMTP connections for test systems.

Before network traffic can enter or leave your EC2 instance, it goes through a firewall provided by AWS. The firewall inspects the network traffic and uses rules to decide whether the traffic is allowed or denied.

IP vs. IP address

The abbreviation *IP* is used for Internet Protocol, whereas an *IP address* describes a specific address like 84.186.116.47.

Figure 6.6 shows how an SSH request from a source IP address 10.0.0.10 is inspected by the firewall and received by the destination IP address 10.10.0.20. In this case, the firewall allows the request because there is a rule that allows TCP traffic on port 22 between the source and the destination.

Source versus destination

Inbound security-group rules filter traffic based on its source. The source is either an IP address or a security group. Thus you can allow inbound traffic only from specific source IP address ranges.

Outbound security-group rules filter traffic based on its destination. The destination is either an IP address or a security group. You can allow outbound traffic to only specific destination IP address ranges.

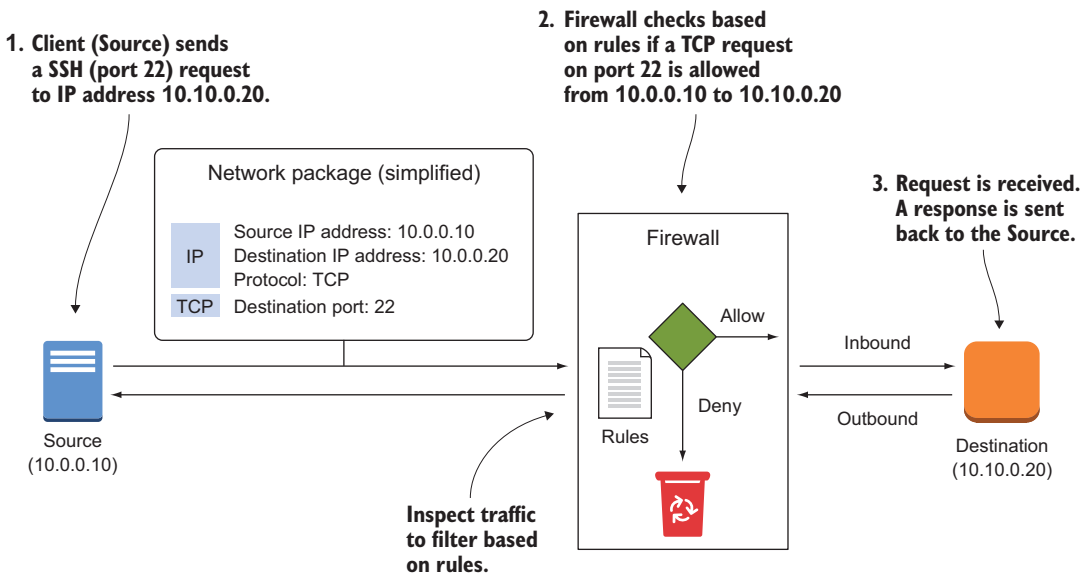


Figure 6.6 How an SSH request travels from source to destination, controlled by a firewall

AWS is responsible for the firewall, but you're responsible for the rules. By default, a security group does not allow any inbound traffic. You must add your own rules to allow specific incoming traffic. A security group contains a rule allowing all outbound traffic by default. If your use case requires a high level of networking security, you should remove the rule and add your own rules to control outgoing traffic.

Debugging or monitoring network traffic

Imagine the following problem: Your EC2 instance does not accept SSH traffic as you want it to, but you can't spot any misconfiguration in your firewall rules. In this case, you should enable VPC Flow Logs to get access to aggregated log messages containing rejected connections. Go to the VPC Flow Logs (AWS Documentation) at http://mng.bz/ltgm to learn more.

VPC Flow Logs give you insight into both granted connections and rejected connections.

6.4.1 Controlling traffic to virtual machines with security groups

Associate a security group with AWS resources such as EC2 instances to control traffic. It's common for EC2 instances to have more than one security group associated with them, and for the same security group to be associated with multiple EC2 instances.

A security group consists of a set of rules. Each rule allows network traffic based on the following:

- Direction (inbound or outbound)
- IP protocol (TCP, UDP, ICMP)
- Port
- Source/destination based on IP address, IP address range, or security group (works only within AWS)

In theory, you could define rules that allow all traffic to enter and leave your virtual machine; AWS won't prevent you from doing so. But it's best practice to define your rules so they are as restrictive as possible.

Security group resources in CloudFormation are of type `AWS::EC2::SecurityGroup`. The following listing is in /chapter06/firewall1.yaml in the book's code folder: the template describes an empty security group associated with a single EC2 instance.

Listing 6.2 CloudFormation template: security group

```
---
[..]
Parameters:
  KeyName:
    Description: 'Key Pair name'
    Type: 'AWS::EC2::KeyPair::KeyName'
    Default: mykey
  VPC:
```

You'll learn about this in section 6.5.

```
    [..]
  Subnet:
    [..]
Resources:
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'Learn how to protect your EC2 Instance.'
      VpcId: !Ref VPC
      Tags:
      - Key: Name
        Value: 'AWS in Action: chapter 6 (firewall)'
  Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-6057e21a'
      InstanceType: 't2.micro'
      KeyName: !Ref KeyName
      NetworkInterfaces:
      - AssociatePublicIpAddress: true
        DeleteOnTermination: true
        DeviceIndex: 0
        GroupSet:
        - !Ref SecurityGroup
        SubnetId: !Ref Subnet
      Tags:
      - Key: Name
        Value: 'AWS in Action: chapter 6 (firewall)'
```

You'll learn about this in section 6.5.

Defines the security group without any rules (by default, inbound traffic is denied and outbound traffic is allowed.) Rules will be added in the following sections.

Defines the EC2 instance

Associates the security group with the EC2 instance

To explore security groups, you can try the CloudFormation template located at http://mng.bz/fVu5. Create a stack based on that template by clicking on the CloudFormation Quick-Create link (http://mng.bz/Qk5e), and then copy the `PublicName` from the stack output.

6.4.2 Allowing ICMP traffic

If you want to ping an EC2 instance from your computer, you must allow inbound Internet Control Message Protocol (ICMP) traffic. By default, all inbound traffic is blocked. Try `ping` *$PublicName* to make sure `ping` isn't working:

```
$ ping ec2-52-5-109-147.compute-1.amazonaws.com
PING ec2-52-5-109-147.compute-1.amazonaws.com (52.5.109.147): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
[...]
```

You need to add a rule to the security group that allows inbound traffic, where the protocol equals ICMP. Listing 6.3 is in /chapter06/firewall2.yaml in the book's code folder.

Listing 6.3 CloudFormation template: security group that allows ICMP

```
SecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Learn how to protect your EC2 Instance.'
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'AWS in Action: chapter 6 (firewall)'
    # allowing inbound ICMP traffic
    SecurityGroupIngress:
    - IpProtocol: icmp
      FromPort: '-1'
      ToPort: '-1'
      CidrIp: '0.0.0.0/0'
```

Rules allowing incoming traffic

Specifies ICMP as the protocol

ICMP does not use ports. -1 means every port.

Allow traffic from any source IP address.

Update the CloudFormation stack with the template located at http://mng.bz/0caa and retry the `ping` command. It should work now:

```
$ ping ec2-52-5-109-147.compute-1.amazonaws.com
PING ec2-52-5-109-147.compute-1.amazonaws.com (52.5.109.147): 56 data bytes
64 bytes from 52.5.109.147: icmp_seq=0 ttl=49 time=112.222 ms
64 bytes from 52.5.109.147: icmp_seq=1 ttl=49 time=121.893 ms
[...]
round-trip min/avg/max/stddev = 112.222/117.058/121.893/4.835 ms
```

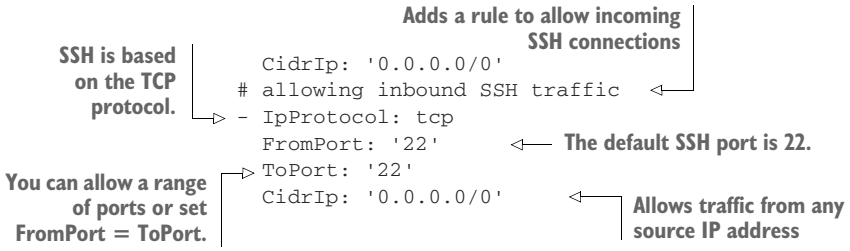
Everyone's inbound ICMP traffic (every source IP address) is now allowed to reach your EC2 instance.

6.4.3 Allowing SSH traffic

Once you can ping your EC2 instance, you want to log in to your virtual machine via SSH. To do so, you must create a rule to allow inbound TCP requests on port 22.

Listing 6.4 CloudFormation template: security group that allows SSH

```
SecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Learn how to protect your EC2 Instance.'
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'AWS in Action: chapter 6 (firewall)'
    # allowing inbound ICMP traffic
    SecurityGroupIngress:
    - IpProtocol: icmp
      FromPort: '-1'
      ToPort: '-1'
```



Update the CloudFormation stack with the template located at http://mng.bz/P8cm. You can now log in to your EC2 instance using SSH. Keep in mind that you still need the correct private key. The firewall only controls the network layer; it doesn't replace key-based or password-based authentication.

6.4.4 Allowing SSH traffic from a source IP address

So far, you're allowing inbound traffic on port 22 (SSH) from every source IP address. It is possible to restrict access to only your own IP address for additional security as well.

> **What's the difference between public and private IP addresses?**
>
> On my local network, I'm using private IP addresses that start with 192.168.0.*. My laptop uses 192.168.0.10, and my iPad uses 192.168.0.20. But if I access the internet, I have the same public IP address (such as 79.241.98.155) for my laptop and iPad. That's because only my internet gateway (the box that connects to the internet) has a public IP address, and all requests are redirected by the gateway. (If you want to know more about this, search for *network address translation*.) Your local network doesn't know about this public IP address. My laptop and iPad only know that the internet gateway is reachable under 192.168.0.1 on the private network.
>
> To find your public IP address, visit http://api.ipify.org. For most of us, our public IP address changes from time to time, usually when you reconnect to the internet (which happens every 24 hours in my case).

Hard-coding the public IP address into the template isn't a good solution because your public IP address changes from time to time. But you already know the solution: parameters. You need to add a parameter that holds your current public IP address, and you need to modify the `Security Group`. You can find the following listing in /chapter06/firewall4.yaml in the book's code folder.

Listing 6.5 Security group allows traffic from source IP

```
Parameters:
  # [...]
  IpForSSH:                          <-- Public IP address parameter
    Description: 'Your public IP address to allow SSH access'
    Type: String
```

```
Resources:
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'Learn how to protect your EC2 Instance.'
      VpcId: !Ref VPC
      Tags:
      - Key: Name
        Value: 'AWS in Action: chapter 6 (firewall)'
      # allowing inbound ICMP traffic
      SecurityGroupIngress:
      - IpProtocol: icmp
        FromPort: '-1'
        ToPort: '-1'
        CidrIp: '0.0.0.0/0'
      # allowing inbound SSH traffic
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        CidrIp: !Sub '${IpForSSH}/32'
```

Uses $IpForSSH/32 as a value

Update the CloudFormation stack with the template located at http://mng.bz/S2f9. When asked for parameters, type in your public IP address for *$IPForSSH*. Now only your IP address can open SSH connections to your EC2 instance.

> **Classless Inter-Domain Routing (CIDR)**
>
> You may wonder what `/32` means in listing 6.5. To understand what's going on, you need to switch your brain into binary mode. An IP address is 4 bytes or 32 bits long. The `/32` defines how many bits (32, in this case) should be used to form a range of addresses. If you want to define the exact IP address that is allowed, you must use all 32 bits.
>
> But sometimes it makes sense to define a range of allowed IP addresses. For example, you can use `10.0.0.0/8` to create a range between 10.0.0.0 and 10.255.255.255, `10.0.0.0/16` to create a range between 10.0.0.0 and 10.0.255.255, or `10.0.0.0/24` to create a range between 10.0.0.0 and 10.0.0.255. You aren't required to use the binary boundaries (8, 16, 24, 32), but they're easier for most people to understand. You already used `0.0.0.0/0` to create a range that contains every possible IP address.

Now you can control network traffic that comes from outside a virtual machine or goes outside a virtual machine by filtering based on protocol, port, and source IP address.

6.4.5 Allowing SSH traffic from a source security group

It is possible to control network traffic based on whether the source or destination belongs to a specific security group. For example, you can say that a MySQL database can only be accessed if the traffic comes from your web servers, or that only your proxy servers are allowed to access the web servers. Because of the elastic nature of the cloud, you'll likely deal with a dynamic number of virtual machines, so rules based on

source IP addresses are difficult to maintain. This becomes easy if your rules are based on source security groups.

To explore the power of rules based on a source security group, let's look at the concept of a *bastion host* for SSH access (some people call it a *jump box*). The trick is that only one virtual machine, the bastion host, can be accessed via SSH from the internet (it should be restricted to a specific source IP address). All other virtual machines can only be reached via SSH from the bastion host. This approach has two advantages:

- You have only one entry point into your system, and that entry point does nothing but SSH. The chances of this box being hacked are small.
- If one of your virtual machines that's running a web server, mail server, FTP server, and so on is hacked, the attacker can't jump from that machine to all the other machines.

To implement the concept of a bastion host, you must follow these two rules:

- Allow SSH access to the bastion host from 0.0.0.0/0 or a specific source address.
- Allow SSH access to all other virtual machines only if the traffic source is the bastion host.

Figure 6.7 shows a bastion host with two EC2 instances that are only reachable via SSH from the bastion host.

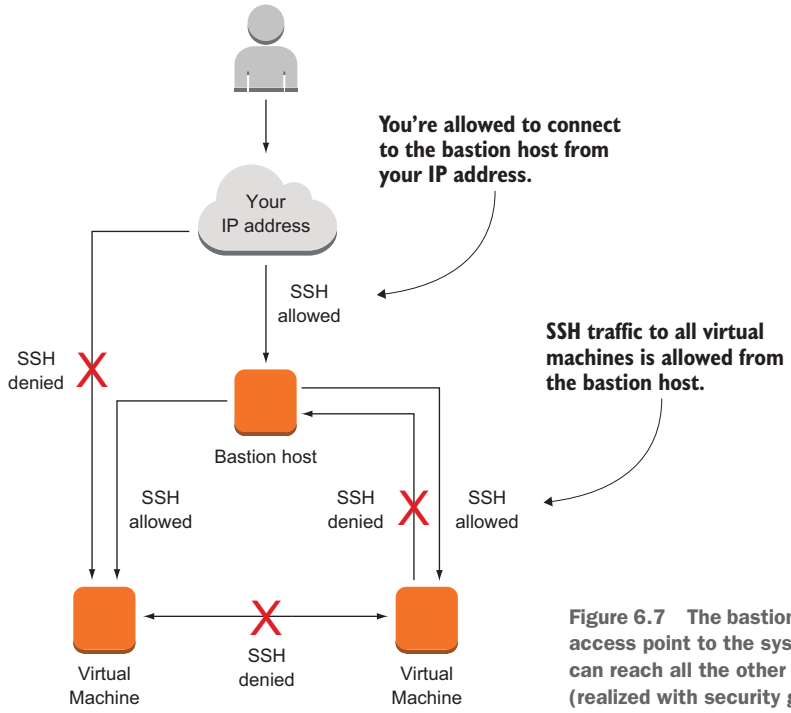


Figure 6.7 The bastion host is the only SSH access point to the system from which you can reach all the other machines via SSH (realized with security groups).

A security group allowing incoming SSH traffic from anywhere needs to be attached to the bastion host. All other VMs are attached to a security group allowing SSH traffic only if the source is the bastion host's security group. The following listing shows the security groups defined in a CloudFormation template:

Listing 6.6 CloudFormation template: SSH from bastion host

```
SecurityGroupBastionHost:                          <-- Security group attached
  Type: 'AWS::EC2::SecurityGroup'                      to the bastion host
  Properties:
    GroupDescription: 'Allowing incoming SSH and ICPM from anywhere.'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - IpProtocol: icmp
      FromPort: "-1"
      ToPort: "-1"
      CidrIp: '0.0.0.0/0'
    - IpProtocol: tcp
      FromPort: '22'
      ToPort: '22'                                 <-- Security group
      CidrIp: !Sub '${IpForSSH}/32'                    attached to another
SecurityGroupInstance:                                 virtual machine
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Allowing incoming SSH from the Bastion Host.'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: '22'                                                <-- Allowing incoming
      ToPort: '22'                                                      SSH traffic only
      SourceSecurityGroupId: !Ref SecurityGroupBastionHost              from bastion host
```

Update the CloudFormation stack with the template located at http://mng.bz/VrWk. If the update is completed, the stack will show three outputs:

1. `BastionHostPublicName`—Use the bastion host to connect via SSH from your computer.
2. `Instance1PublicName`—You can connect to this EC2 instance only from the bastion host.
3. `Instance2PublicName`—You can connect to this EC2 instance only from the bastion host.

Execute the following command to add your key to the SSH agent. Replace *`$PathToKey`* with the path to the SSH key:

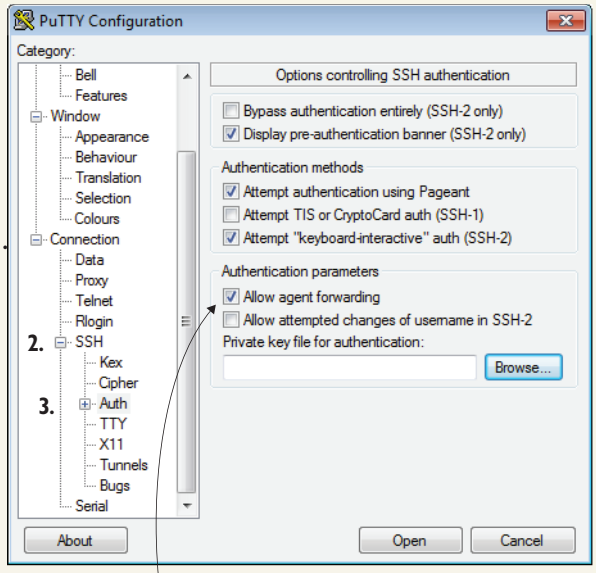
```
ssh-add $PathToKey/mykey.pem
```

Now connect to `BastionHostPublicName` via SSH. Replace *`$BastionHostPublicName`* with the public name of the bastion host.

```
ssh -A ec2-user@$BastionHostPublicName
```

Agent forwarding with PuTTY

To make agent forwarding work with PuTTY, you need to make sure your key is loaded into PuTTY Pageant by double-clicking the private key file. You must also enable Connection > SSH > Auth > Allow Agent Forwarding.



The `-A` option is important for enabling `AgentForwarding`; agent forwarding lets you authenticate with the same key you used to log in to the bastion host for further SSH logins initiated from the bastion host.

Log in to `$Instance1PublicName` or `$Instance2PublicName` from the bastion host next.

```
[computer]$ ssh -A ec2-user@ec2-52-4-234-102.[...].com
Last login: Sat Apr 11 11:28:31 2015 from [...]
[...]
[bastionh]$ ssh ec2-52-4-125-194.compute-1.amazonaws.com
Last login: Sat Apr 11 11:28:43 2015 from [...]
[...]
```

Log in to the bastion host.

Log in to $Instance1PublicName from the bastion host.

The bastion host can be used to add a layer of security to your system. If one of your virtual machines is compromised, the attacker can't jump to other machines in your system. This reduces the potential damage the attacker can inflict. It's important that the bastion host does nothing but SSH, to reduce the chance of it becoming a security problem. We use the bastion-host pattern frequently to protect our clients' infrastructure.

Using agent forwarding is a security risk

We are using agent forwarding in our examples when establishing a SSH connection from the bastion host to one of the other two instances. Agent forwarding is a potential security risk, because the bastion host is able to read the private key from your computer. Therefore, you need to fully trust the bastion host when using agent forwarding.

A more secure alternative is using the bastion host as a proxy. The following command establishes an SSH connection to instance 1 by using the bastion host as a proxy.

```
ssh -J ec2-user@BastionHostPublicName ec2-user@Instance1PublicName
```

In this case the bastion host does not need to access the private key and you can disable agent forwarding.

Cleaning up

Don't forget to delete your stack after you finish this section, to clean up all used resources. Otherwise you'll likely be charged for the resources you use.

6.5 Creating a private network in the cloud: Amazon Virtual Private Cloud (VPC)

When you create a VPC, you get your own private network on AWS. *Private* means you can use the address ranges 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16 to design a network that isn't necessarily connected to the public internet. You can create subnets, route tables, ACLs, and gateways to the internet or a VPN endpoint .

Subnets allow you to separate concerns. Create a separate subnet for your databases, web servers, proxy servers, or application servers, or whenever you can separate two systems. Another rule of thumb is that you should have at least two subnets: public and private. A public subnet has a route to the internet; a private subnet doesn't. Your load balancer or web servers should be in the public subnet, and your database should reside in the private subnet.

For the purpose of understanding how a VPC works, you'll create a VPC to host an enterprise web application. You'll re-implement the bastion host concept from the previous section by creating a public subnet that contains only the bastion host server. You'll also create a private subnet for your web servers and one public subnet for your proxy servers. The proxy servers absorb most of the traffic by responding with the latest version of the page they have in their cache, and they forward traffic to the private web servers. You can't access a web server directly over the internet—only through the web caches.

The VPC uses the address space 10.0.0.0/16. To separate concerns, you'll create two public subnets and one private subnet in the VPC:

- 10.0.1.0/24 public SSH bastion host subnet
- 10.0.2.0/24 public Varnish proxy subnet
- 10.0.3.0/24 private Apache web server subnet

> **What does 10.0.0.0/16 mean?**
> 10.0.0.0/16 represents all IP addresses in 10.0.0.0 and 10.0.255.255. It's using CIDR notation (explained earlier in the chapter).

Network ACLs restrict traffic that goes from one subnet to another, acting as a firewall. That's an additional layer of security on top of security groups, which control traffic to and from a virtual machine. The SSH bastion host from section 6.4 can be implemented with these ACLs:

- SSH from 0.0.0.0/0 to 10.0.1.0/24 is allowed.
- SSH from 10.0.1.0/24 to 10.0.2.0/24 is allowed.
- SSH from 10.0.1.0/24 to 10.0.3.0/24 is allowed.

To allow traffic to the Varnish proxy and the Apache web servers, you'll need these additional ACLs:

- HTTP from 0.0.0.0/0 to 10.0.2.0/24 is allowed.
- HTTP from 10.0.2.0/24 to 10.0.3.0/24 is allowed.

Figure 6.8 shows the architecture of the VPC.

You'll use CloudFormation to describe the VPC with its subnets. The template is split into smaller parts to make it easier to read in the book. As usual, you'll find the code in the book's code repository on GitHub: https://github.com/AWSinAction/code2. The template is located at /chapter06/vpc.yaml.

6.5.1 Creating the VPC and an internet gateway (IGW)

The first resources listed in the template are the VPC and the internet gateway (IGW). The IGW will translate the public IP addresses of your virtual machines to their private IP addresses using network address translation (NAT). All public IP addresses used in the VPC are controlled by this IGW:

```
VPC:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: '10.0.0.0/16'
    EnableDnsHostnames: 'true'
    Tags:
    - Key: Name
      Value: 'AWS in Action: chapter 6 (VPC)'
```

The IP address space used for the private network

Adds a Name tag to the VPC

```
InternetGateway:
  Type: 'AWS::EC2::InternetGateway'
  Properties: {}
VPCGatewayAttachment:
  Type: 'AWS::EC2::VPCGatewayAttachment'
  Properties:
    VpcId: !Ref VPC
    InternetGatewayId: !Ref InternetGateway
```

An IGW is needed to enable traffic to and from the internet.

Attaches the internet gateway to the VPC

Next you'll define the subnet for the bastion host.

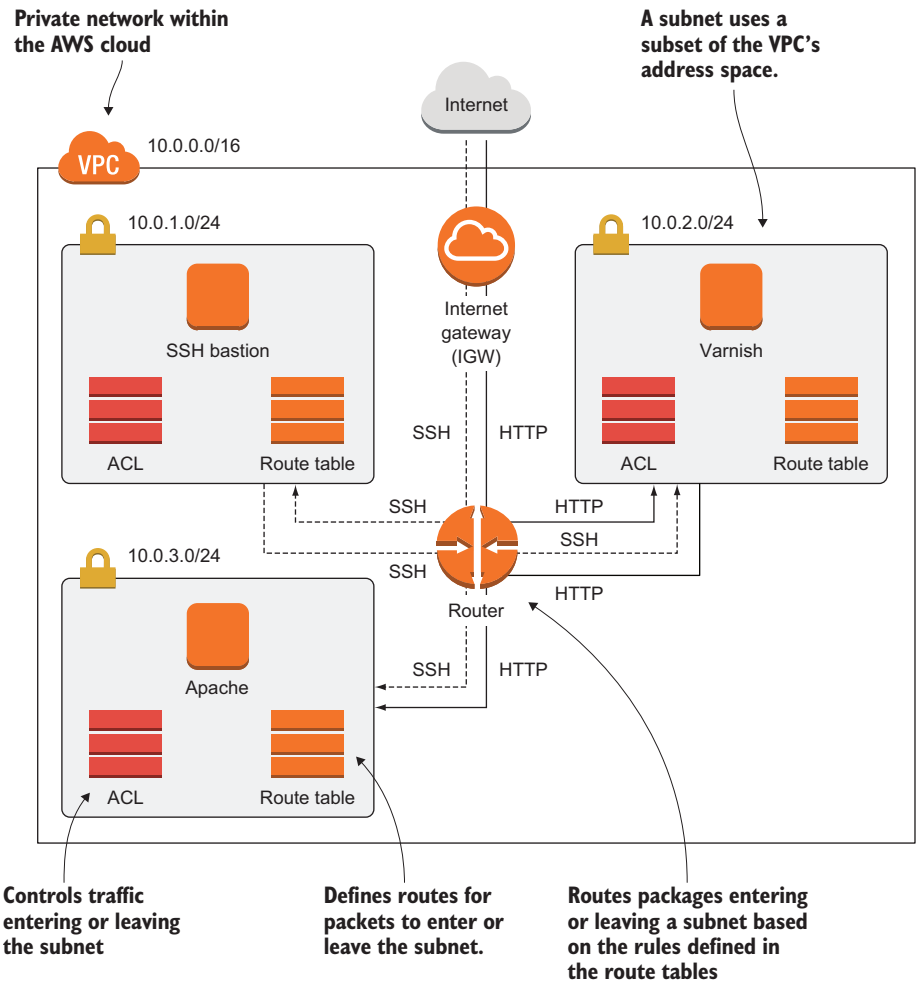


Figure 6.8 VPC with three subnets to secure a web application

6.5.2 Defining the public bastion host subnet

The bastion host subnet will only run a single machine to secure SSH access:

```
SubnetPublicBastionHost:
  Type: 'AWS::EC2::Subnet'
  Properties:
    AvailabilityZone: 'us-east-1a'
    CidrBlock: '10.0.1.0/24'
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'Public Bastion Host'
RouteTablePublicBastionHost:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
RouteTableAssociationPublicBastionHost:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref SubnetPublicBastionHost
    RouteTableId: !Ref RouteTablePublicBastionHost
RoutePublicBastionHostToInternet:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref RouteTablePublicBastionHost
    DestinationCidrBlock: '0.0.0.0/0'
    GatewayId: !Ref InternetGateway
  DependsOn: VPCGatewayAttachment
NetworkAclPublicBastionHost:
  Type: 'AWS::EC2::NetworkAcl'
  Properties:
    VpcId: !Ref VPC
SubnetNetworkAclAssociationPublicBastionHost:
  Type: 'AWS::EC2::SubnetNetworkAclAssociation'
  Properties:
    SubnetId: !Ref SubnetPublicBastionHost
    NetworkAclId: !Ref NetworkAclPublicBastionHost
```

You'll learn about this in chapter 11.

IP address space

Route table

Associates the route table with the subnet

Routes everything (0.0.0.0/0) to the IGW

Network ACL

Associates the NACL with the subnet

The definition of the ACL follows:

```
NetworkAclEntryInPublicBastionHostSSH:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicBastionHost
    RuleNumber: '100'
    Protocol: '6'
    PortRange:
      From: '22'
      To: '22'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryInPublicBastionHostEphemeralPorts:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicBastionHost
```

Allows inbound SSH from everywhere

Use the rule number to define the order of rules.

Inbound

Ephemeral ports used for short-lived TCP/IP connections

```
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '10.0.0.0/16'
NetworkAclEntryOutPublicBastionHostSSH:       <-- Allows outbound SSH to VPC
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicBastionHost
    RuleNumber: '100'
    Protocol: '6'
    PortRange:
      From: '22'
      To: '22'
    RuleAction: 'allow'
    Egress: 'true'              <-- Outbound
    CidrBlock: '10.0.0.0/16'
NetworkAclEntryOutPublicBastionHostEphemeralPorts: <-- Ephemeral ports
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicBastionHost
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
```

There's an important difference between security groups and ACLs: security groups are stateful, but ACLs aren't. If you allow an inbound port on a security group, the corresponding response to requests on that port are allowed as well. A security group rule will work as you expect it to. If you open inbound port 22 on a security group, you can connect via SSH.

That's not true for ACLs. If you open inbound port 22 on an ACL for your subnet, you still may not be able to connect via SSH. In addition, you need to allow outbound ephemeral ports, because sshd (SSH daemon) accepts connections on port 22 but uses an ephemeral port for communication with the client. Ephemeral ports are selected from the range starting at 1024 and ending at 65535.

If you want to make a SSH connection from within your subnet, you have to open outbound port 22 and inbound ephemeral ports as well.

There is another difference between security group rules and ACL rules: you have to define the priority for ACL rules. A smaller rule number indicates a higher priority. When evaluating an ACL the first rule that matches a package is applied; all other rules are skipped.

We do recommend to start with using security groups to control traffic. If you want to add an extra layer of security, you should use ACL on top.

6.5.3 Adding the private Apache web server subnet

The subnet for the Varnish web cache is similar to the bastion host subnet because it's also a public subnet; that's why we'll skip it. You'll continue with the private subnet for the Apache web server:

```
SubnetPrivateApacheWebserver:
  Type: 'AWS::EC2::Subnet'
  Properties:
    AvailabilityZone: 'us-east-1a'
    CidrBlock: '10.0.3.0/24'          <-- Address space
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'Private Apache Webserver'
RouteTablePrivateApacheWebserver:     <-- No route to the IGW
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
RouteTableAssociationPrivateApacheWebserver:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref SubnetPrivateApacheWebserver
    RouteTableId: !Ref RouteTablePrivateApacheWebserver
```

As shown in figure 6.9, the only difference between a public and a private subnet is that a private subnet doesn't have a route to the IGW.

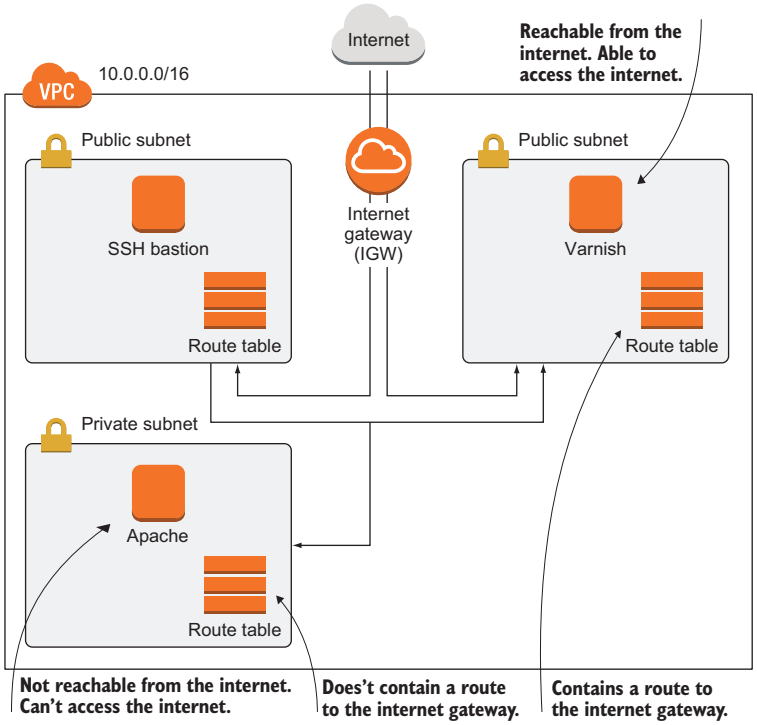


Figure 6.9 Private and public subnets

Traffic between subnets of a VPC is always routed by default. You can't remove the routes between the subnets. If you want to prevent traffic between subnets in a VPC, you need to use ACLs attached to the subnets.

6.5.4 Launching virtual machines in the subnets

Your subnets are ready, and you can continue with the EC2 instances. First you describe the bastion host:

```
BastionHost:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: 'ami-6057e21a'
    InstanceType: 't2.micro'
    KeyName: mykey
    NetworkInterfaces:
    - AssociatePublicIpAddress: true
      DeleteOnTermination: true
      DeviceIndex: '0'
      GroupSet:
      - !Ref SecurityGroup
      SubnetId: !Ref SubnetPublicBastionHost
    Tags:
    - Key: Name
      Value: 'Bastion Host'
  DependsOn: VPCGatewayAttachment
```

Assigns a public IP address

This security group allows everything.

Launches in the bastion host subnet

The Varnish proxy server looks similar. But again, the private Apache web server has a different configuration:

```
ApacheWebserver:
  Type: 'AWS::EC2::Instance'
  Properties:
    ImageId: 'ami-6057e21a'
    InstanceType: 't2.micro'
    KeyName: mykey
    NetworkInterfaces:
    - AssociatePublicIpAddress: false
      DeleteOnTermination: true
      DeviceIndex: '0'
      GroupSet:
      - !Ref SecurityGroup
      SubnetId: !Ref SubnetPrivateApacheWebserver
    UserData:
      'Fn::Base64': !Sub |
        #!/bin/bash -x
        bash -ex << "TRY"
          yum -y install httpd
          service httpd start
        TRY
        /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} \
    ➥ --resource ApacheWebserver --region ${AWS::Region}
    Tags:
```

No public IP address: private

Launches in the Apache web server subnet

If one of the following two commands fails, the sub-bash stops at this command and returns a non zero exit code.

Installs Apache from the internet

Starts Apache web server

```
    - Key: Name
      Value: 'Apache Webserver'
  CreationPolicy:
    ResourceSignal:
      Timeout: PT10M
  DependsOn: RoutePrivateApacheWebserverToInternet
```

You're now in serious trouble: installing Apache won't work because your private subnet has no route to the internet.

6.5.5 Accessing the internet from private subnets via a NAT gateway

Public subnets have a route to the internet gateway. You can use a similar mechanism to provide internet access for private subnets without having a direct route to the internet: use a NAT gateway in a public subnet, and create a route from your private subnet to the NAT gateway. This way, you can reach the internet from private subnets, but the internet can't reach your private subnets. A NAT gateway is a managed service provided by AWS that handles network address translation. Internet traffic from your private subnet will access the internet from the public IP address of the NAT gateway.

> **Reducing costs for NAT gateway**
>
> You have to pay for the traffic processed by a NAT gateway (see VPC Pricing at https://aws.amazon.com/vpc/pricing/ for more details). If your EC2 instances in private subnets will have to transfer huge amounts of data to the Internet, there are two options to decrease costs.
>
> - Moving your EC2 instances from the private subnet to a public subnet allows them to transfer data to the internet without utilizing the NAT gateway. Use firewalls to strictly restrict incoming traffic from the internet.
> - If data is transferred over the internet to reach AWS services (such as Amazon S3 and Amazon DynamoDB), use so-called VPC endpoints. These endpoints allow your EC2 instances to communicate with S3 and DynamoDB directly without using the NAT gateway. Furthermore, some services are accessible from private subnets via AWS PrivateLink (such as Amazon Kinesis, AWS SSM, and more). Note: AWS PrivateLink is not yet available in all regions.

To keep concerns separated, you'll create a subnet for the NAT gateway.

```
SubnetPublicNAT:
  Type: 'AWS::EC2::Subnet'
  Properties:
    AvailabilityZone: 'us-east-1a'
    CidrBlock: '10.0.0.0/24'
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'Public NAT'
```

10.0.0.0/24 is the NAT subnet.

```
RouteTablePublicNAT:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
# [...]
RoutePublicNATToInternet:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref RouteTablePublicNAT
    DestinationCidrBlock: '0.0.0.0/0'
    GatewayId: !Ref InternetGateway
  DependsOn: VPCGatewayAttachment
# [...]
EIPNatGateway:
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: 'vpc'
NatGateway:
  Type: 'AWS::EC2::NatGateway'
  Properties:
    AllocationId: !GetAtt 'EIPNatGateway.AllocationId'
    SubnetId: !Ref SubnetPublicNAT
# [...]
RoutePrivateApacheWebserverToInternet:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref RouteTablePrivateApacheWebserver
    DestinationCidrBlock: '0.0.0.0/0'
    NatGatewayId: !Ref NatGateway
```

The NAT subnet is public with a route to the internet.

A static public IP address is used for the NAT gateway.

The NAT gateway is placed into the private subnet and associated with the static public IP address.

Route from the Apache subnet to the NAT gateway

WARNING The NAT gateway included in the example is not covered by the Free Tier. The NAT gateway will cost you $0.045 USD per hour and $0.045 per GB of data processed when creating the stack in the US East (N. Virginia) region. Go to https://aws.amazon.com/vpc/pricing/ to have a look at the current prices.

Now you're ready to create the CloudFormation stack with the template located at http://mng.bz/NRLj by clicking on the CloudFormation Quick-Create Link: http://mng.bz/71o2. Once you've done so, copy the `VarnishProxyPublicName` output and open it in your browser. You'll see an Apache test page that was cached by Varnish.

Cleaning up

Don't forget to delete your stack after finishing this section, to clean up all used resources. Otherwise you'll likely be charged for the resources you use.

Summary

- AWS is a shared-responsibility environment in which security can be achieved only if you and AWS work together. You're responsible for securely configuring your AWS resources and your software running on EC2 instances, while AWS protects buildings and host systems.
- Keeping your software up-to-date is key, and can be automated.
- The Identity and Access Management (IAM) service provides everything needed for authentication and authorization with the AWS API. Every request you make to the AWS API goes through IAM to check whether the request is allowed. IAM controls who can do what in your AWS account. To protect your AWS account, grant only those permissions that your users and roles need.
- Traffic to or from AWS resources like EC2 instances can be filtered based on protocol, port, and source or destination.
- A bastion host is a well-defined single point of access to your system. It can be used to secure SSH access to your virtual machines. Implementation can be done with security groups or ACLs.
- A VPC is a private network in AWS where you have full control. With VPCs, you can control routing, subnets, ACLs, and gateways to the internet or your company network via VPN. A NAT gateway enables access to the internet from private subnets.
- You should separate concerns in your network to reduce potential damage if, for example, one of your subnets is hacked. Keep every system in a private subnet that doesn't need to be accessed from the public internet, to reduce your attackable surface.